Negotiation-Based Resource Provisioning and Task Scheduling Algorithm for Cloud Systems

Ji Li[1], Yanzhi Wang[2], Xue Lin[1], Shahin Nazarian[1] and Massoud Pedram[1]

[1]Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
[2]College of Engineering and Computer Science, Syracuse University, Syracuse, NY, USA
jli724@usc.edu, ywang393@syr.edu, {xuelin, shahin, pedram}@usc.edu

***Abstract*—Cloud computing has recently drawn significant attention of both academia and industry as an emerging computing paradigm where data, applications, or processing power are provided as services through the Internet connection. Cloud computing extends the existing computing infrastructure owned by the cloud service providers (CSPs) to achieve the economies of scale through virtualization and aggregated computing resources. End users, on the other hand, can reach these services through an elastic utility computing environment with minimal upfront investment. Nevertheless, pervasive use of cloud computing and the resulting rise in the number of data centers have brought forth the concerns about energy consumption and carbon emission. Therefore, this paper addresses the problem of resource provisioning and task scheduling on a cloud platform under the service level agreement, in order to minimize the electric bills and maximize the profitability for the CSP. User task graphs and dependencies are randomly generated, whereas user requests for CPU and memory resources are extracted from Google cluster trace. A general type of dynamic pricing scenario is assumed where the energy price is both time-of-use and total power consumption-dependent. A negotiation-based iterative approach has been proposed for the resource provisioning and task scheduling that is inspired by a Field-Programmable Gate Array (FPGA) routing algorithm. More specifically, in each iteration, decisions made in the previous iteration are ripped-up and re-decided, and the concept of congestion is introduced to dynamically adjust the resource provisioning decisions and the schedule of each task based on the historical results as well as the current status. Experimental results demonstrate that the proposed algorithm achieves up to 63.52% improvement in the total energy price compared to baseline.**

I. Introduction

Cloud computing is an emerging computing paradigm in which hosted software, platforms and infrastructure are delivered as services through pooled resources, geographic diversity and universal connectivity [1]. Virtualization, which is recognized as the fundamental technology that powers cloud computing, enables cloud service providers (CSPs) to make more efficient use of hardware resources such as data centers and server clusters through facilitating a greater degree of abstraction of the software environment [1], [2]. With virtualization technology, end users, who are incentivized by the savings of reducing or eliminating the costs associated with hardware, software or licensing fees, can rent a variety of virtual machines (VMs) from CSP based on the requirements of their own applications (jobs/tasks). On the other hand, CSP achieves economies of scale and makes profits by charging a time-based or usage-based fee from a large number of end users. There are many cloud computing platforms in the market such as Google App Engine (GAE) [3], Amazon Elastic Compute Cloud (EC2) [4], and Microsoft Windows Azure Platform [5].

Service-Level Agreement (SLA) is a service contract negotiated and established among CSP and end users, which documents privacy, security, and the agreed-upon levels of availability, serviceability, compensation in the event of a contract breach, and performance metrics like deadline (delivery time of service). CSP should consistently meet the agreements written in the SLA and meanwhile aim to increase market share, growth and profitability through reducing operational expenses. It is widely known that the bulk of the CSP operating cost comes from the electric bills of data centers where large groups of disks, servers, and routers are networked. In the U.S., data centers consumed an estimated 91 billion kWhs of electricity in 2013, equivalent to the annual output of 34 large (500 megawatt) coal-fired power plants [6]. As a further matter, data center electricity consumption is projected to increase to roughly 140 billion kWhs annually by 2020, costing companies 13 billion U.S. dollars annually in electric bills and emitting nearly 100 million metric tons of carbon pollution per year [6]. This necessitates the development of efficient management techniques for CSPs to reduce the energy consumption in data centers.

According to [7], most of today's servers have two important energy usage features: (i) servers tend to be significantly more inefficient under low levels of utilization than being exercised at the optimal utilization rate (around 70-80% depending on the servers), and (ii) servers may have significant power consumption during idle periods. Therefore, server consolidation and load balancing can be applied to achieve a high performance-to-power ratio through selectively shutting down idle servers and improving the utilization levels in active servers. Of note, with various advances in server energy conservation technologies, high-end modern servers are able to cut down the active idle power to less than 20% of their peak power consumption under peak load condition [8]. Nevertheless, such servers are extremely expensive and the majority of servers in data centers are previous models or latest low/medium-end models, which are still suffering from the energy inefficiency caused by server idle power consumption.

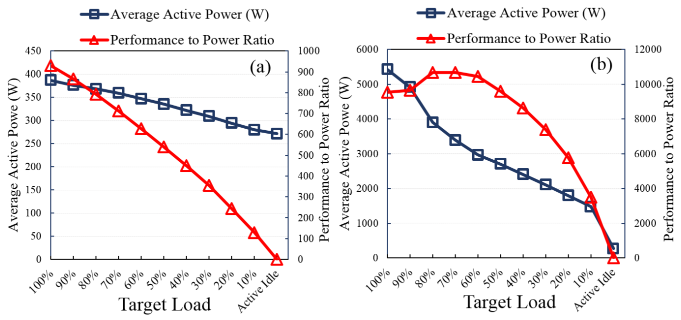


Fig. 1: An example benchmark result for SPECpower_ssj2008: (a) typical low/medium-end model: Hewlett-Packard Company Proliant DL580 G5 and (b) typical modern high-end model: Dell Inc. PowerEdge M830

In this sense, server consolidation and load balancing are more attractive solutions than huge investments in high-end servers. Figure 1 (a) and (b) provide the energy usage profiles for a typical low/medium-end server and a latest high-end server model, respectively. The high-end model is much more efficient than the low/medium-end model during idle or low utilization periods, whereas the price of the high-end model is about 28X of the low/medium-end model.

Considerable research efforts have been conducted in the context of improving energy efficiency in cloud data centers using server consolidation and load balancing. The authors in [9] propose a hybrid particle swarm optimization with an improved ant colony algorithm to achieve a good trade-off between the rate of convergence and the scale of global search space, however, workloads and cloud platform are assumed to be homogeneous. To consider heterogeneous characteristics in both workloads and servers, the authors in [10] propose a heterogeneity-aware resource management system for dynamic capacity provisioning in cloud computing environments. In order to achieve the overall efficiency with the awareness of other competing users and the entire cloud resource map, the authors in [11] develop a genetic algorithm which deals with resource provisioning, VM placement, and task scheduling in a holistic fashion. However, the genetic algorithm in [11] have scalability problem.

In this paper, we consider the resource provisioning and task scheduling problem for a CSP that owns a number of heterogeneous server farms, in order to minimize the CSP's electric bills under dynamic pricing. A general type of dynamic pricing policy is considered in this paper, which is comprised of a time-of-use (TOU) price and a power consumption-dependent price. Users request VMs using the Pay-As-You-Go option (which is available in most cloud computing platforms such as Amazon EC2 [4] and Microsoft Azure [5]), and the workloads of users are modelled as a collection of multiple job graphs with internal dependencies. The server farms comprise a diverse mix of computing and data resources, and the entire cloud platform is modelled as a weighted graph where communication among servers incurs extra delay. In the Pay-As-You-Go agreement, users do not need to schedule tasks or architect data storage. Instead, CSP determines the scheduling as well as the allocation of physical resources to host the VMs such that the overall efficiency is maximized while stipulated requirements in SLA are met. Figure 2 provides the overview of the cloud computing environment. The workloads sent by users will first go through an *admission control policy* which filters the jobs that are unable to meet deadlines even with the best available computing resources allocated. Next, CSP, which is assumed to have abundant VM resources to accommodate users' requests, will configure VMs for the remaining tasks and schedule the tasks using a resource provisioning and task scheduling algorithm.

In order to solve the aforementioned problem efficiently, we develop a novel negotiation-based cloud resource provisioning and task scheduling algorithm for CSP to fully leverage the flexible, scalable infrastructure and cost savings. The proposed algorithm adopts an iterative manner and is inspired by the negotiation-based FPGA routing method in [12], and this heuristic method has also been applied to other task scheduling problems [13], [14]. In order to balance the load through the entire scheduling period, the concept of congestion is introduced, which guides the scheduler to shift loads from high utilization time periods to low utilization periods. In each iteration, the proposed algorithm rips-up all the decisions that are made in the previous iteration, re-decides VM configuration for each task and re-schedules all tasks based on the degree of congestion. Congestion related terms are trained in each iteration and guide the scheduler to achieve overall energy efficiency gradually. Job graphs and task dependencies are randomly generated and real-world user workload requests for CPU and memory resources provided in Google cluster trace are used to conduct the experiments [15]. Experimental results demonstrate that the proposed algorithm can achieve up to 63.52% energy price reduction compared to baseline.

II. System Model

A. Time Model, Price Model and Workload Model

In this paper, we adopt a *slotted time* model where all system parameters, constraints and scheduling decisions are provided for discrete time intervals of constant and equal length, which is denoted by T_{unit}. The proposed algorithm can achieve near-continuous results with fine-grained time slots.

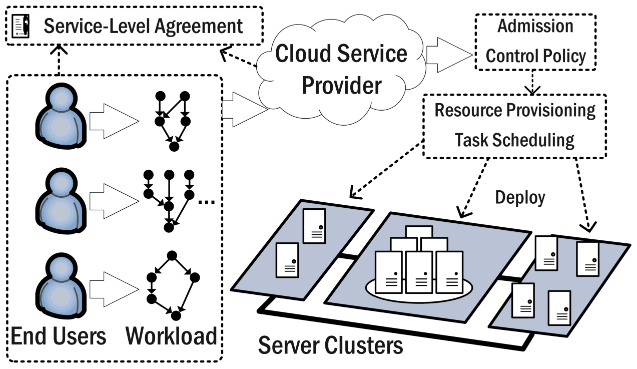


Fig. 2: System model of the cloud platform

End users send N job requests to the CSP, where each job a $(1 \leq a \leq N)$ consists of a total number of n_a tasks with dependencies. Job a is represented by a directed acyclic graph (DAG) $G_a(V_a, E_a)$, where each vertex T_i^a $(1 \leq i \leq n_a)$ embodies a task. Of note, each task can be uniquely indexed by T_i^a, and the total number of tasks in all jobs is denoted by $n_{total} = \Sigma_{a=1}^{N} n_a$. Edge from task T_i^a to task T_j^a indicates that T_j^a depends on the output of T_i^a and the amount of data that needs to be fed into T_j^a from T_i^a is represented by the weight of the edge $W_{i,j}^a$. The entire workload request can be represented by a set of N DAGs. Figure 3 illustrates an example of user workload. End users also specify a deadline $L_{deadline}^a$ for each job a, and we use $L_i^a(g)$ to denote the worst case execution time of task T_i^a with g type VM allocated. The analysis of worst case execution time affects the results but falls outside the scope of this paper. For each task T_i^a, the scheduled start time is denoted by s_i^a. The entire scheduling period is denoted by L_{max}, which equals the last finish time among all the tasks. A combined price model $\epsilon\big(t, P_{total}(t)\big)$ is considered in this paper, which is comprised of a TOU price determined by the time slot t and a power consumption-dependent price that depends on the total instantaneous power consumption $P_{total}(t)$ in the time slot t. The TOU price is higher in peak usage time slots than off-peak time slots, which incentivizes energy users to shift loads towards off-peak periods. The power consumption-dependent price is assumed to be monotonically increasing with respect to $P_{total}(t)$, which penalizes peak energy consumption that can potentially affect grid stability and lead to blackouts, power cuts, and brownouts.

The cloud platform supports K categories of VMs where tasks are executed. Each type VM_g is associated with a two-tuple parameter set $\{R_{MEM}^g, R_{CPU}^g\}$, which represents the amount of memory and CPU resources required to establish a VM_g on a server. Each task T_i^a is also associated with a two-tuple parameter set $\{F_{MEM}^{a,i}, F_{CPU}^{a,i}\}$, which indicates the minimum amount of memory and CPU resources required to run task T_i^a. Therefore, task T_i^a can only be executed in those types of VMs with abundant resources (i.e., $F_{MEM}^{a,i} \leq R_{MEM}^g$ and $F_{CPU}^{a,i} \leq R_{CPU}^g$), and we use an integer set θ_i^a to represent all the types of VMs that support task T_i^a. Accordingly, end users can only request the VM type g that support the requested task T_i^a, i.e., $\forall\ T_i^a$, requested VM type $g \in \theta_i^a$.

B. Cloud Platform Model

The cloud infrastructure of the CSP is comprised of several server clusters located in different data center campuses, and the entire cloud platform is modelled as an undirected graph $G_{CSP}(D, B)$. There are M vertices in the vertex set D, which denote the M servers $(D_1, D_2, ..., D_M)$ in the cloud infrastructure. Each edge $B_{x,y}$ embodies the link between server D_x and D_y, and the communication bandwidth of the link $B_{x,y}$ is represented by its weight $b_{x,y}$. By default, data exchange in the same server does not incur extra delay, i.e., $B_{x,x} = \infty$, whereas communication among different servers results in transmission overhead. Figure 4 provides an example of cloud platform, which consists of communication links and two server clusters, each with six servers. As mentioned in section I, we consider heterogeneous server farms in this paper. Each server D_x is associated with a two-tuple parameter set $\{C_{MEM}^x, C_{CPU}^x\}$, which represents the total amount of memory and CPU resources available on the server D_x. Besides, each server D_x can only support several VM types and an integer set ζ_x stores all the types of VMs that server D_x can support. Each server is also associated with a $K \times L_{max}$ VM configuration matrix $\mathbf{Q_x}$, where element $Q^x(g,t)$ at the g-th row and t-th column indicates the number of g type VMs allocated on the server D_x in the t-th time slot. Obviously, the following constraints must be satisfied all the time: (i) $\forall g \notin \zeta_x$ and $\forall t \in [1, L_{max}]$, $Q^x(g,t) = 0$, and (ii) $\forall t$ and $\forall x$, $\Sigma_{g=1}^{K} Q^x(g,t) \cdot R_{CPU}^g \leq C_{CPU}^x$ and $\Sigma_{g=1}^{K} Q^x(g,t) \cdot R_{MEM}^g \leq C_{MEM}^x$.

C. Energy Consumption Model and Problem Formulation

The energy consumption in each server D_x depends on the *utilization rate* $U_x(t)$, which is the ratio of the CPU usage in time slot t over the total CPU resources available on server D_x. Hence, the utilization rate $U_x(t)$ is calculated as

$$U_x(t) = \frac{\Sigma_{g=1}^{K} Q^x(g,t) \cdot R_{CPU}^g}{C_{CPU}^x} \times 100\% \tag{1}$$

During operation, each server D_x has static power consumption $P_{static}^x(t)$ and dynamic power consumption $P_{dynamic}^x(t)$, and the total power consumption $P_{total}^x(t)$ is calculated as

$$P_{total}^x(t) = P_{static}^x(t) + P_{dynamic}^x(t) \tag{2}$$

$P_{static}^x(t)$ is constant when $U_x(t) > 0$, 0 otherwise. $P_{dynamic}^x(t)$, on the other hand, is related to the optimal utilization level of server D_x, which is denoted by Opt_x. The Opt_x of most modern servers is about 70-80%. Dynamic power consumption increases drastically when utilization level is greater than the optimal utilization rate. We adopt the

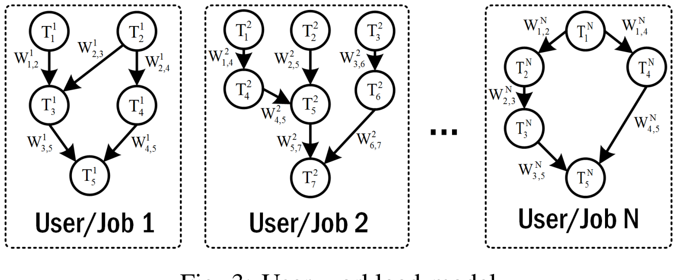


Fig. 3: User workload model

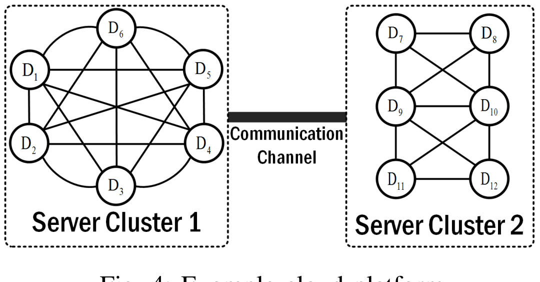


Fig. 4: Example cloud platform

dynamic power consumption model developed in [11] and $P^x_{dynamic}(t)$ is calculated as

$$\begin{cases} \alpha_x \cdot U_x(t), & \text{if } U_x(t) < Opt_x \\ \alpha_x \cdot Opt_x + \beta_x \cdot \big(U_x(t) - Opt_x\big)^2, & \text{otherwise} \end{cases} \tag{3}$$

where α_x and β_x are server dependent parameters. Using the abovementioned models, the resource provisioning and task scheduling problem is formulated as follows:

Resource Provisioning and Task Scheduling (RPTS) Problem for CSP.

Given price model, workload and cloud platform information.

Find the VM configuration and start time for each task.

Minimize the total energy price during the entire scheduling period:

$$Total\ Cost = \sum_{t=1}^{L_{max}} \epsilon\bigg(t, \sum_{x=1}^{M} \Big(P^x_{total}(t)\Big)\bigg) \tag{4}$$

Subject to:

$$\Sigma^K_{g=1} Q^x(g,t) \cdot R^g_{CPU} \le C^x_{CPU},\ \forall t \text{ and } \forall a \tag{5}$$

$$\Sigma^K_{g=1} Q^x(g,t) \cdot R^g_{MEM} \le C^x_{MEM},\ \forall t \text{ and } \forall a \tag{6}$$

$$s^a_i + L^a_i(g) \le L^a_{deadline}, \forall T^a_i \tag{7}$$

$$F^{a,i}_{MEM} \le R^g_{MEM}, \forall T^a_i \tag{8}$$

$$F^{a,i}_{CPU} \le R^g_{CPU}, \forall T^a_i \tag{9}$$

$$Q^x(g,t) = 0, \forall g \notin \zeta_x \text{ and } \forall t \in [1, L_{max}] \tag{10}$$

III. Negotiation-Based Resource Provisioning and Task Scheduling Algorithm

The RPTS problem described in Section II-C is an integer non-linear programming problem subject to integer non-linear constraints. No provably optimal solutions can be derived in polynomial time, nor are they likely to ever be available for this kind of problems [16]. Therefore, in this paper, we invest an efficient heuristic to search for a suitable solution instead of the optimal one. The remainder of this section is organized as follows. Subsection III-A explains the motivation to introduce the negotiation-based routing method to the RPTS problem. Subsection III-B presents the proposed negotiation-based resource provisioning and task scheduling algorithm.

A. Motivation to Introduce Negotiation-Based Routing Method to the Resource Provisioning and Task Scheduling Problem

The FPGA routing process aims to connect signals properly according to the gate-level logic descriptions such that the worst case delay is minimized and the route is feasible, i.e., no signals share any physical wire or switch. In order to solve the FPGA routing problem efficiently, the authors in [17] proposed the *Negotiated Congestion* (NC) algorithm, where the concept of congestion is introduced to represent the degree of routing resource sharing. More specifically, in the NC algorithm, several congestion terms are integrated into the objective function (i.e., path delay) and the router is guided by these terms to avoid congested wires or switches while aiming to minimize the path delay. There are certain similarities between the FPGA routing problem and the RPTS problem:

- Sharing routing resources in FPGA among signals leads to infeasible routing solution, and utilizing the same time slots among tasks on the same server result in high utilization and low energy efficiency.
- In both problems, finding the optimal solution is impractical due to the extremely long runtime. Hence, it is reasonable to seek for a suitable solution instead of the optimal one in both problems.
- Considering the large scale of search space in both problems, one pass is not sufficient to generate a good enough solution. Therefore, iterative approach is more suitable for both problems.

Motivated by the above similarities, we introduce the concept of congestion to the proposed algorithm to effectively solve the RPTS problem.

B. Negotiation-Based Resource Provisioning and Task Scheduling Algorithm

In this subsection, a *Negotiation-Based Resource Provisioning and Task Scheduling* (NBRPTS) algorithm is proposed to effectively solve the RPTS problem described in Section II-C.

As mentioned in Section I, an iterative approach is adopted. The proposed algorithm finds a good enough solution when the total cost in Eqn. (4) has not decreased for d_{term} iterations or the iteration number exceeds the maximum iteration number denoted by d_{max}. The results of the algorithm are the *configuration* and *scheduling* of each task. The configuration determines the VM type and physical server, and the scheduling is the start time to execute the task. We define a *ready task* as a task with all its dependencies satisfied. In each iteration, all tasks T^a_i $(1 \le a \le N, 1 \le i \le n_a)$ will be ripped-up, i.e., decisions made in the previous iteration will be cleaned, and the VM configuration as well as the scheduling for each ready task will be re-decided one by one until all tasks are re-configured and re-scheduled. We denote the configuration and scheduling decision in the d-th iteration after scheduling the f-th task $(1 \le f \le n_{total})$ by S^d_f. Accordingly, static and dynamic power consumption during the configuration and scheduling process are functions of the configuration and schedule decision, which are represented by $P^x_{static}(t, S^d_f)$ and $P^x_{dynamic}(t, S^d_f)$, respectively. Similarly, the total power consumption of server D_x becomes $P^x_{total}(t, S^d_f)$. We define the *Cost Increase* (CI) after configuring and scheduling the f-th task in the d-th iteration as follows

$$CI = \sum_{t=1}^{L_{max}} \Bigg(\epsilon\bigg(t, \sum_{x=1}^{M} \Big(P^x_{total}(t, S^d_f)\Big)\bigg) - \epsilon\bigg(t, \sum_{x=1}^{M} \Big(P^x_{total}(t, S^d_{f-1})\Big)\bigg)\Bigg) \tag{11}$$

Clearly, for each ready task, the S_f^d with the minimum CI will be chosen.

One important observation is that it is more likely to see a high utilization rate and high energy price in the time slots which have already been occupied by other tasks. Such time slots are considered as *congested*, and it is reasonable to avoid congested time slots in the rest of the iteration in order to minimize the total energy price. An *intra-iteration congestion term* $R(t)$ is introduced to quantify the degree of congestion, which is equal to the total number of tasks that have been scheduled to occupy time slot t within an iteration. Another important observation is that tasks configured earlier may have the same configuration decisions (i.e., occupy the same energy efficient VMs and servers) in all iterations. This may block other configurations scenarios where tasks configured later take the energy efficient VMs as well as servers and the overall energy cost is lower. Therefore, we introduce an *inter-iteration congestion term* $h_i^a(x,g)$ to indicate the total number of times in the previous iterations when task T_i^a is allocated to server D_x with VM_g. The term $h_i^a(x,g)$ gradually increases if task T_i^a is allocated to the same server and VM in several iterations, and eventually task T_i^a will be guided to explore other configuration possibilities.

In addition to the abovementioned congestion terms, we introduce another *inter-iteration term* $V_a(t)$ to represent the total number of times when job a violates the deadline requirement in Eqn. (7) at the t-th time slot in the previous iterations. The time slots that are occupied by job a and cause deadline violation in the previous iteration will see higher energy costs as the term $V_a(t)$ gradually increases. Hence, the term $V_a(t)$ penalizes deadline violation and incentivizes tasks of job a to utilize time slots that do not violate the deadline requirement. With the aforementioned terms, in the d-th iteration, the CI after configuring and scheduling the f-th task T_i^a is modified as:

$$CI' = \sum_{t=1}^{L_{max}} \Bigg(\epsilon\Big(t, \sum_{x=1}^{M} \Big(P_{total}^x(t, S_f^d)\Big)\Big) - \epsilon\Big(t, \sum_{x=1}^{M} \Big(P_{total}^x(t, S_{f-1}^d)\Big)\Big)\Bigg) \cdot \Big(1 + c_1 \cdot R(t)\Big) \cdot \Big(1 + c_2 \cdot \sum_{g=1}^{K}\sum_{x=1}^{M} h_i^a(x,g)\Big) + \sum_{t=1}^{L_{max}} c_3 \cdot V_a(t) \quad (12)$$

where c_1, c_2, and c_3 are positive weights of $R(t)$, $h_i^a(x,g)$, and $V_a(t)$, respectively. After introducing the above terms, the NBRPTS algorithm will avoid congested time slots within an iteration due to the term $c_1 \cdot R(t)$, and explore other configuration possibilities since the term $h_i^a(x,g)$ permanently increases when task T_i^a is configured onto server D_x using the g type VM after several iterations. If tasks in job a occupy time slots which violate the deadline requirement, the term $V_a(t)$ will increase the energy costs and after several iterations, tasks in job a will give up these time slots due to the high energy costs caused by the term $V_a(t)$ and try to complete execution in early time slots.

Initially, all the terms $R(t)$, $h_i^a(x,g)$, and $V_a(t)$ are zeros, and the CI' is the same as CI. Intra-iteration term $R(t)$ is reset at the beginning of each iteration and updated after each decision S_f^d is made. Inter-iteration terms $h_i^a(x,g)$ and $V_a(t)$, on the other hand, are reset only during initiation and updated at the end of each iteration. The computational complexity of the proposed algorithm is $O(d_{max}L_{max}n_{total}^2MK)$. The n_{total}^2 term comes from finding each ready task and for each ready task, conducting configuration as well as scheduling. Algorithm 1 provides the pseudo code of the proposed algorithm.

Algorithm 1: Negotiation-Based Resource Provisioning and Task Scheduling Algorithm

```
1  Initialize task graphs, price model, cloud platform model and terms;
2  Set iteration counter $d = 0$;
3  repeat
4      $d = d + 1$;
5      foreach time slot $t$ do $R(t) = 0$;
6      foreach task do
7          rip up all configuration and scheduling decisions;
8      end
9      repeat
10         Find a ready task $T_i^a$;
11         Configure and schedule the task $T_i^a$ such that $CI'$ in Eqn. (12) is minimized;
12         Update the intra-iteration term $R(t)$;
13     until all tasks are configured and scheduled;
14     foreach job $a$ do
15         foreach time slot $t$ do
16             Update inter-iteration term $V_a(t)$;
17             foreach server $D_x$ do
18                 foreach VM type $g$ do
19                     Update inter-iteration term $h_i^a(x,g)$;
20                 end
21             end
22         end
23     end
24     Calculate total energy price in Eqn. (4);
25 until total energy cost has not decreased for $d_{term}$ iterations or $d > d_{max}$;
26 return configuration and scheduling of each task;
```

IV. Experimental Results

In this section, we demonstrate the effectiveness of the proposed algorithm on a heterogeneous cloud platform, which consists of up to 4 server farms and 30 heterogeneous servers. Communication among servers in the same server farm is much faster than communication among servers in different server farms. We randomly generate user task graphs with different numbers of tasks in each job, and the dependencies among tasks in a job are represented using a DAG. In order to consider the dynamics of the jobs and resource requests arriving at the cluster, we extract the vector of resource request per task in terms of CPU and memory from the Google cluster dataset released in 2011 [15], which is measured on a heterogeneous server cluster during a 29-day period. The

CPU and memory resource profiles of VMs are generated such that each VM can support only a subset of tasks, whereas the resources available on each server are provided so that each server can accommodate a limited number of VMs simultaneously. Table I summarizes the key parameters of the cloud platform and workloads.

TABLE I: Modeling parameters summary

Cloud Platform Parameters		User Workload Characteristics		
Servers Farms	Servers	Total Users	Task per User	Task Latency
2-4	15-30	5-20	5-14	2-7

We assume the utility company provides the TOU-dependent price as well as the instantaneous power consumption-dependent price, and the power consumption-dependent price part is monotonically increasing with the CSP's real-time total power consumption. We compare the proposed algorithm with a greedy baseline that schedules each ready task to start at the best possible time slot based on the cost increase function in Eqn. (11).

We first compare the total energy cost through the entire 29-day period (i.e., different resource request) using a fixed job number $N = 10$ and a fixed cloud platform with 4 server farms and 30 servers. Results in Figure 5 demonstrate that the proposed NBRPTS algorithm consistently outperforms the baseline during the entire 29-day period. Next, using the resource request information on the first day, we sweep the number of jobs from 5 to 20 on the same cloud platform and sweep the server number from 10 to 30 using the same job number, in order to show the effects of workloads and server resources, respectively. Due to space limitation, we only show the normalized energy cost results in Figure 6. Of note, less energy cost in larger number of jobs, as shown in Figure 6 (a), is caused by the randomness of task and platform generation. The proposed NBRPTS algorithm achieves up to 63.52% energy cost savings compared to baseline with different job numbers and server numbers. Besides, the proposed algorithm results in deadline violations only in a few scenarios in the previous experiments, whereas the baseline has deadline violations in most cases (because the baseline algorithm does not consider deadline). The worst case runtime of the proposed algorithm is 152.62s in the previous experiments.

V. Conclusion

The overarching goal of this paper was to minimize the total energy cost for CSP under dynamic pricing, while meeting the deadline requirements in the SLA. An iterative negotiation-based resource provisioning and task scheduling algorithm was proposed, where the concept of congestion was introduced, in order to guide the scheduler to avoid congested time periods and escape local optimum traps. The experimental results demonstrated that the proposed algorithm could achieve up to 63.52% improvement compared to the baseline.

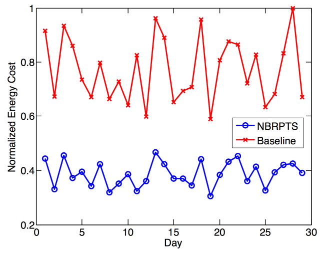
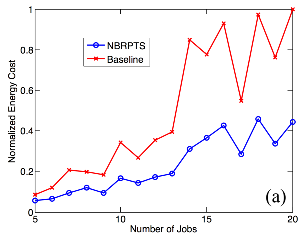


Fig. 5: Impact of task resource request



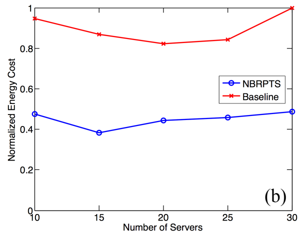


Fig. 6: (a) Impact of job number and (b) impact of server resources

VI. Acknowledgement

This work is sponsored in part by a grant from the National Science Foundation.

References

[1] J. Hoffman. Cloud computing: Take your virtual machines to the cloud. [Online]. Available: https://technet.microsoft.com/en-us/magazine/hh126814.aspx
[2] S. Angeles. Virtualization vs. cloud computing: What's the difference? [Online]. Available: http://www.businessnewsdaily.com/5791-virtualization-vs-cloud-computing.html
[3] Google app engine. [Online]. Available: https://cloud.google.com/appengine/docs
[4] Amazon web services. [Online]. Available: http://aws.amazon.com
[5] Microsoft azure. [Online]. Available: http://azure.microsoft.com
[6] P. Delforge. America's data centers consuming and wasting growing amounts of energy. [Online]. Available: http://www.nrdc.org/energy/data-center-efficiency-assessment.asp
[7] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.
[8] Specpower ssj2008 results. [Online]. Available: https://www.spec.org
[9] Y. Xiaoguang, C. Tingbin, and Z. Qisong, "Research on cloud computing schedule based on improved hybrid pso," in *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*. IEEE, 2013, pp. 388–391.
[10] Q. Zhang *et al.*, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud."
[11] Y. Gao *et al.*, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2013, p. 31.
[12] J. Luu *et al.*, "Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, p. 32, 2011.
[13] J. Li *et al.*, "Negotiation-based task scheduling to minimize users electricity bills under dynamic energy prices," 2014.
[14] ——, "Negotiation-based task scheduling and storage control algorithm to minimize user's electric bills under dynamic prices," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*.
[15] Google cluster data. [Online]. Available: https://github.com/google/cluster-data
[16] R. Hemmecke *et al.*, "Nonlinear integer programming," *arXiv preprint arXiv:0906.5171*, 2009.
[17] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for fpgas," in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*. ACM, 1995, pp. 111–117.